Principal Component Analysis

Mark Richardson

May 2009

Contents

1 Introduction

Principal Component Analysis (PCA) is the general name for a technique which uses sophisticated underlying mathematical principles to transforms a number of possibly correlated variables into a smaller number of variables called principal components. The origins of PCA lie in multivariate data analysis, however, it has a wide range of other applications, as we will show in due course. PCA has been called, 'one of the most important results from applied linear algebra'[2] and perhaps its most common use is as the first step in trying to analyse large data sets. Some of the other common applications include; de-noising signals, blind source separation, and data compression.

In general terms, PCA uses a vector space transform to reduce the dimensionality of large data sets. Using mathematical projection, the original data set, which may have involved many variables, can often be interpreted in just a few variables (the principal components). It is therefore often the case that an examination of the reduced dimension data set will allow the the user to spot trends, patterns and outliers in the data, far more easily than would have been possible without performing the principal component analysis.

The aim of this essay is to explain the theoretical side of PCA, and to provide examples of its application. We will begin with a non-rigorous motivational example from multivariate data analysis in which we will attempt to extract some meaning from a 17 dimensional data set. After this motivational example, we shall discuss the PCA technique in terms of its linear algebra fundamentals. This will lead us to a method for implementing PCA for real-world data, and we will see that there is a close connection between PCA and the singular value decomposition (SVD) from numerical linear algebra. We will then look at two further examples of PCA in practice; Image Compression and Blind Source Separation.

2 An Example From Multivariate Data Analysis

In this section, we will examine some real life multivariate data in order to explain, in simple terms what PCA achieves. We will perform a principal component analysis of this data and examine the results, though we will skip over the computational details for now.

Suppose that we are examining the following DEFRA[1] data showing the consumption in grams (per person, per week) of 17 different types of foodstuff measured and averaged in the four countries of the United Kingdom in 1997. We shall say that the 17 food types are the *variables* and the 4 countries are the *observations*. A cursory glance over the numbers in Table 1 does not reveal much, indeed in general it is difficult to extract meaning from any given array of numbers. Given that this is actually a relatively small data set, we see that a powerful analytical method is absolutely necessary if we wish to observe trends and patterns in larger data.

	England	Wales	Scotland	N Ireland
Cheese	105	103	103	66
Carcass meat	245	227	242	267
Other meat	685	803	750	586
Fish	147	160	122	93
Fats and oils	193	235	184	209
Sugars	156	175	147	139
Fresh potatoes	720	874	566	1033
Fresh Veg	253	265	171	143
Other Veg	488	570	418	355
Processed potatoes	198	203	220	187
Processed Veg	360	365	337	334
Fresh fruit	1102	1137	957	674
Cereals	1472	1582	1462	1494
Beverages	57	73	53	47
Soft drinks	1374	1256	1572	1506
Alcoholic drinks	375	475	458	135
Confectionery	54	64	62	41

Table 1: UK food consumption in 1997 (g/person/week). Source: DEFRA website

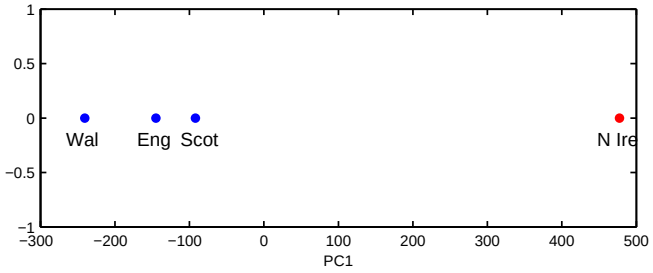
We need some way of making sense of the above data. Are there any trends present which are not obvious from glancing at the array of numbers? Traditionally, we would use a series of bivariate plots (scatter diagrams) and analyse these to try and determine any relationships between variables, however the number of such plots required for such a task is typically $O(n^2)$, where n is the number of variables. Clearly, for large data sets, this is not feasible.

PCA generalises this idea and allows us to perform such an analysis simultaneously, for many variables. In our example above, we have 17 dimensional data for 4 countries. We can thus 'imagine' plotting the 4 coordinates representing the 4 countries in 17 dimensional space. If there is any correlation between the observations (the countries), this will be observed in the 17 dimensional space by the correlated points being clustered close together, though of course since we cannot visualise such a space, we are not able to see such clustering directly.

[1] Department for Environment, Food and Rural Affairs

The first task of PCA is to identify a new set of orthogonal coordinate axes through the data. This is achieved by finding the direction of maximal variance through the coordinates in the 17 dimensional space. It is equivalent to obtaining the (least-squares) line of best fit through the plotted data. We call this new axis the *first principal component* of the data. Once this first principal component has been obtained, we can use orthogonal projection[2] to map the coordinates down onto this new axis. In our food example above, the four 17 dimensional coordinates are projected down onto the first principal component to obtain the following representation in Figure 1.

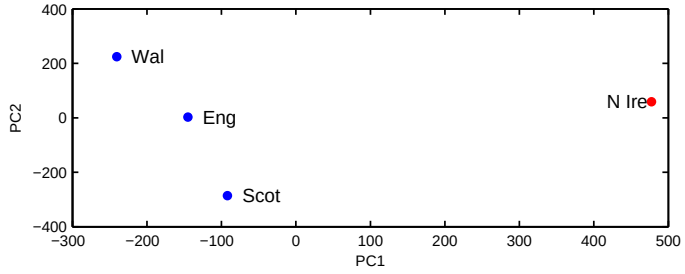
Figure 1: Projections onto first principal component (1-D space)



This type of diagram is known as a *score plot*. Already, we can see that the there are two potential clusters forming, in the sense that England, Wales and Scotland seem to be close together at one end of the principal component, whilst Northern Ireland is positioned at the opposite end of the axis.

The PCA method then obtains a second principal coordinate (axis) which is both orthogonal to the first PC, and is the next best direction for approximating the original data (i.e. it finds the direction of second largest variance in the data, chosen from directions which are orthogonal to the first principal component). We now have two orthogonal principal components defining a plane which, similarly to before, we can project our coordinates down onto. This is shown below in the 2 dimensional score plot in Figure 2. Notice that the inclusion of the second principal component has highlighted variation between the dietary habits present England, Scotland and Wales.

Figure 2: Projections onto first 2 principal components (2-D space)



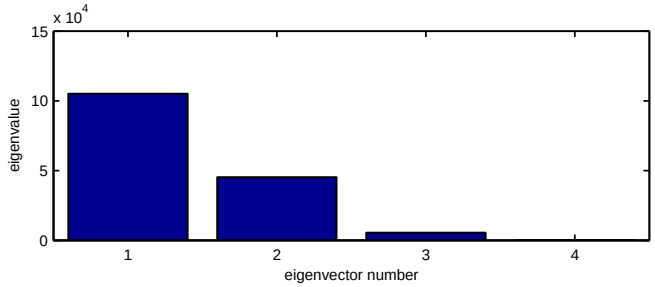
As part of the PCA method (which will be explained in detail later), we automatically obtain information about the contributions of each principal component to the total variance of the coordinates. In fact, in this case approximately 67% of the variance in the data is accounted for by the first principal component, and approximately 97% is accounted for in total by the first two principal components. In this case, we have therefore accounted for the vast majority of the variation in the data using a two dimensional plot - a dramatic reduction in dimensionality from seventeen dimensions to two.

[2]In linear algebra and functional analysis, a *projection* is defined as a linear transformation, P, that maps from a given vector space to the same vector space and is such that $P^2 = P$.

In practice, it is usually sufficient to include enough principal components so that somewhere in the region of $70 - 80\%$ of the variation in the data is accounted for [3].

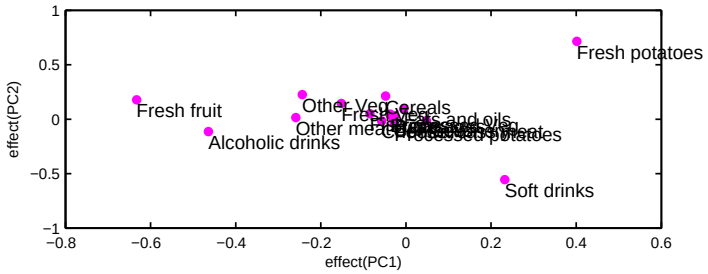
This information can be summarised in a plot of the variances (nonzero eigenvalues) with respect to the principal component number (eigenvector number), which is given in Figure 3, below.

Figure 3: Eigenspectrum



We can also consider the influence of each of the original variables upon the principal components. This information can be summarised in the following plot, in Figure 4.

Figure 4: Load plot



Observe that there is a central group of variables around the middle of each principal component, with four variables on the periphery that do not seem to be part of the group. Recall the 2D score plot (Figure 2), on which England, Wales and Scotland were clustered together, whilst Northern Ireland was the country that was away from the cluster. Perhaps there is some association to be made between the four variables that are away from the cluster in Figure 4 and the country that is located away from the rest of the countries in Figure 2, Northern Ireland. A look at the original data in Table 1 reveals that for the three variables, *Fresh potatoes, Alcoholic drinks* and *Fresh fruit*, there is a noticeable difference between the values for England, Wales and Scotland, which are roughly similar, and Northern Ireland, which is usually significantly higher or lower.

PCA has the ability to be able to make these associations for us. It has also successfully managed to reduce the dimensionality of our data set down from 17 to 2, allowing us to assert (using Figure 2) that countries England, Wales and Scotland are 'similar' with Northern Ireland being different in some way. Furthermore, using Figure 4 we were able to associate certain food types with each cluster of countries.

3 The Technical Details Of PCA

The principal component analysis for the example above took a large set of data and identified an optimal new basis in which to re-express the data. This mirrors the general aim of the PCA method: *can we obtain another basis that is a linear combination of the original basis and that re-expresses the data optimally*? There are some ambiguous terms in this statement, which we shall address shortly, however for now let us frame the problem in the following way.

Assume that we start with a data set that is represented in terms of an $m \times n$ matrix, $\mathbf{X}$ where the n columns are the samples (e.g. observations) and the m rows are the variables. We wish to linearly transform this matrix, $\mathbf{X}$ into another matrix, $\mathbf{Y}$, also of dimension $m \times n$, so that for some $m \times m$ matrix, $\mathbf{P}$,

$$\mathbf{Y} = \mathbf{PX} \tag{1}$$

This equation represents a change of basis. If we consider the *rows* of $\mathbf{P}$ to be the row vectors $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_m}$, and the *columns* of $\mathbf{X}$ to be the column vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$, then (3) can be interpreted in the following way.

$$\mathbf{PX} = \begin{pmatrix} \mathbf{Px_1} & \mathbf{Px_2} & \ldots & \mathbf{Px_n} \end{pmatrix} = \begin{pmatrix} \mathbf{p_1}.\mathbf{x_1} & \mathbf{p_1}.\mathbf{x_2} & \cdots & \mathbf{p_1}.\mathbf{x_n} \\ \mathbf{p_2}.\mathbf{x_1} & \mathbf{p_2}.\mathbf{x_2} & \cdots & \mathbf{p_2}.\mathbf{x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{p_m}.\mathbf{x_1} & \mathbf{p_m}.\mathbf{x_2} & \cdots & \mathbf{p_m}.\mathbf{x_n} \end{pmatrix} = \mathbf{Y}$$

Note that $p_i, x_j \in \mathbb{R}^m$, and so $\mathbf{p_i}.\mathbf{x_j}$ is just the standard Euclidean inner (dot) product. This tells us that the original data, $\mathbf{X}$ is being *projected* on to the columns of $\mathbf{P}$. Thus, the rows of $\mathbf{P}$, $\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_m}\}$ are a new *basis* for representing the columns of X. The rows of $\mathbf{P}$ will later become our principal component directions.

We now need to address the issue of what this new basis should be, indeed what is the 'best' way to re-express the data in $\mathbf{X}$ - in other words, how should we define *independence* between principal components in the new basis?

Principal component analysis defines independence by considering the variance of the data in the original basis. It seeks to de-correlate the original data by finding the directions in which variance is maximised and then use these directions to define the new basis. Recall the definition for the variance of a random variable, Z with mean, μ.

$$\sigma_Z^2 = E[(Z - \mu)^2]$$

Suppose we have a vector of n discrete measurements, $\tilde{\mathbf{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n)$, with mean μ_r. If we subtract the mean from each of the measurements, then we obtain a translated set of measurements $\mathbf{r} = (r_1, r_2, \ldots, r_n)$, that has zero mean. Thus, the variance of these measurements is given by the relation

$$\sigma_{\mathbf{r}}^2 = \frac{1}{n}\mathbf{r}\mathbf{r}^T$$

If we have a second vector of n measurements, $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, again with zero mean, then we can generalise this idea to obtain the *covariance* of $\mathbf{r}$ and $\mathbf{s}$. Covariance can be thought of as a measure of how much two variables change together. Variance is thus a special case of covariance, when the the two variables are identical. It is in fact correct to divide through by a factor of $n - 1$ rather than n, a fact which we shall not justify here, but is discussed in [2].

$$\sigma^2_{\mathbf{rs}} = \frac{1}{n-1}\mathbf{r}\mathbf{s}^T$$

We can now generalise this idea to considering our $m \times n$ data matrix, $\mathbf{X}$. Recall that m was the number of variables, and n the number of samples. We can therefore think of this matrix, X in terms of m *row* vectors, each of length n.

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix} = \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \vdots \\ \mathbf{x_m} \end{pmatrix} \in \mathbb{R}^{m \times n}, \qquad \mathbf{x_i}^T \in \mathbb{R}^n$$

Since we have a row vector for each variable, each of these vectors contains all the samples for one particular variable. So for example, $\mathbf{x_i}$ is a vector of the n samples for the i^{th} variable. It therefore makes sense to consider the following matrix product.

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T = \frac{1}{n-1}\begin{pmatrix} \mathbf{x_1}\mathbf{x_1}^T & \mathbf{x_1}\mathbf{x_2}^T & \cdots & \mathbf{x_1}\mathbf{x_m}^T \\ \mathbf{x_2}\mathbf{x_1}^T & \mathbf{x_2}\mathbf{x_2}^T & \cdots & \mathbf{x_2}\mathbf{x_m}^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x_m}\mathbf{x_1}^T & \mathbf{x_m}\mathbf{x_2}^T & \cdots & \mathbf{x_m}\mathbf{x_m}^T \end{pmatrix} \in \mathbb{R}^{m \times m}$$

If we look closely at the entries of this matrix, we see that we have computed all the possible covariance pairs between the m variables. Indeed, on the diagonal entries, we have the *variances* and on the off-diagonal entries, we have the *covariances*. This matrix is therefore known as the *Covariance Matrix*.

Now let us return to the original problem, that of linearly transforming the original data matrix using the relation $\mathbf{Y} = \mathbf{P}\mathbf{X}$, for some matrix, $\mathbf{P}$. We need to decide upon some features that we would like the transformed matrix, $\mathbf{Y}$ to exhibit and somehow relate this to the features of the corresponding covariance matrix $\mathbf{C_Y}$.

Covariance can be considered to be a measure of how well correlated two variables are. The PCA method makes the fundamental assumption that the variables in the transformed matrix should be as uncorrelated as possible. This is equivalent to saying that the covariances of different variables in the matrix $\mathbf{C_Y}$, should be as close to zero as possible (covariance matrices are always positive definite or positive semi-definite). Conversely, large variance values interest us, since they correspond to interesting dynamics in the system (small variances may well be noise). We therefore have the following requirements for constructing the covariance matrix, $\mathbf{C_Y}$:

1. Maximise the signal, measured by variance (maximise the diagonal entries)
2. Minimise the covariance between variables (minimise the off-diagonal entries)

We thus come to the conclusion that since the minimum possible covariance is zero, we are seeking a *diagonal matrix*, $\mathbf{C_Y}$. If we can choose the transformation matrix, $\mathbf{P}$ in such a way that $\mathbf{C_Y}$ is diagonal, then we will have achieved our objective.

We now make the assumption that the vectors in the new basis, $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_m}$ are orthogonal (in fact, we additionally assume that they are orthonormal). Far from being restrictive, this assumption enables us to proceed by using the tools of linear algebra to find a solution to the problem. Consider the formula for the covariance matrix, $\mathbf{C_Y}$ and our interpretation of $\mathbf{Y}$ in terms of $\mathbf{X}$ and $\mathbf{P}$.

$$\mathbf{C_Y} = \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T = \frac{1}{n-1}(\mathbf{PX})(\mathbf{PX})^T = \frac{1}{n-1}(\mathbf{PX})(\mathbf{X}^T\mathbf{P}^T) = \frac{1}{n-1}\mathbf{P}(\mathbf{X}\mathbf{X}^T)\mathbf{P}^T$$

$$\text{i.e.} \qquad \mathbf{C_Y} = \frac{1}{n-1}\mathbf{P}\mathbf{S}\mathbf{P}^T \qquad \text{where} \qquad \mathbf{S} = \mathbf{X}\mathbf{X}^T$$

Note that $\mathbf{S}$ is an $m \times m$ symmetric matrix, since $(\mathbf{X}\mathbf{X}^T)^T = (\mathbf{X}^T)^T(\mathbf{X})^T = \mathbf{X}\mathbf{X}^T$. We now invoke the well known theorem from linear algebra that every square symmetric matrix is orthogonally (orthonormally) diagonalisable. That is, we can write:

$$\mathbf{S} = \mathbf{E}\mathbf{D}\mathbf{E}^T$$

Where $\mathbf{E}$ is an $m \times m$ orthonormal matrix whose columns are the orthonormal eigenvectors of $\mathbf{S}$, and $\mathbf{D}$ is a diagonal matrix which has the eigenvalues of $\mathbf{S}$ as its (diagonal) entries. The rank, r, of $\mathbf{S}$ is the number of orthonormal eigenvectors that it has. If $\mathbf{B}$ turns out to be rank-deficient so that r is less than the size, m, of the matrix, then we simply need to generate $m - r$ orthonormal vectors to fill the remaining columns of $\mathbf{S}$.

It is at this point that we make a choice for the transformation matrix, $\mathbf{P}$. By choosing the *rows* of $\mathbf{P}$ to be the eigenvectors of $\mathbf{S}$, we ensure that $\mathbf{P} = \mathbf{E}^T$ and vice-versa. Thus, substituting this into our derived expression for the covariance matrix, $\mathbf{C_Y}$ gives:

$$\begin{aligned} \mathbf{C_Y} &= \frac{1}{n-1}\mathbf{P}\mathbf{S}\mathbf{P}^T \\ &= \frac{1}{n-1}\mathbf{E}^T(\mathbf{E}\mathbf{D}\mathbf{E}^T)\mathbf{E} \end{aligned}$$

Now, since $\mathbf{E}$ is an orthonormal matrix, we have $\mathbf{E}^T\mathbf{E} = \mathbf{I}$, where $\mathbf{I}$ is the $m \times m$ identity matrix. Hence, for this special choice of $\mathbf{P}$, we have:

$$\mathbf{C_Y} = \frac{1}{n-1}\mathbf{D}$$

A last point to note is that with this method, we automatically gain information about the relative importance of each principal component from the variances. The largest variance corresponds to the first principal component, the second largest to the second principal component, and so on. This therefore gives us a method for organising the data in the diagonalisation stage. Once we have obtained the eigenvalues and eigenvectors of $\mathbf{S} = \mathbf{X}\mathbf{X}^T$, we sort the eigenvalues in descending order and place them in this order on the diagonal of $\mathbf{D}$. We then construct the orthonormal matrix, $\mathbf{E}$ by placing the associated eigenvectors in the same order to form the columns of $\mathbf{E}$ (i.e. place the eigenvector that corresponds to the largest eigenvalue in the first column, the eigenvector corresponding to the second largest eigenvalue in the second column etc.).

We have therefore achieved our objective of diagonalising the covariance matrix of the transformed data. The principal components (the rows of $\mathbf{P}$) are the eigenvectors of the covariance matrix, $\mathbf{X}\mathbf{X}^T$, and the rows are in order of 'importance', telling us how 'principal' each principal component is.

4 The Singular Value Decomposition

In this section, we will examine how the well known singular value decomposition (SVD) from linear algebra can be used in principal component analysis. Indeed, we will show that the derivation of PCA in the previous section and the SVD are closely related. We will not derive the SVD, as it is a well established result, and can be found in any good book on numerical linear algebra, such as [4].

Given $\mathbf{A} \in \mathbb{R}^{n \times m}$, not necessarily of full rank, a singular value decomposition of $\mathbf{A}$ is:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

Where

$$\begin{aligned} \mathbf{U} &\in \mathbb{R}^{n \times n} && \text{is orthonormal} \\ \mathbf{\Sigma} &\in \mathbb{R}^{n \times m} && \text{is diagonal} \\ \mathbf{V} &\in \mathbb{R}^{m \times m} && \text{is orthonormal} \end{aligned}$$

In addition, the diagonal entries, σ_i, of Σ are non-negative and are called the *singular values* of $\mathbf{A}$. They are ordered such that the largest singular value, σ_1 is placed in the $(1, 1)$ entry of Σ, and the other singular values are placed in order down the diagonal, and satisfy $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_p \geq 0$, where $p = \min(n, m)$. Note that we have reversed the row and column indexes in defining the SVD from the way they were defined in the derivation of PCA in the previous section. The reason for doing this will become apparent shortly.

The SVD can be considered to be a general method for understanding change of basis, as can be illustrated by the following argument (which follows [4]).

Since $\mathbf{U} \in \mathbb{R}^{n \times n}$ and $\mathbf{V} \in \mathbb{R}^{m \times m}$ are orthonormal matrices, their columns form bases for, respectively, the vector spaces $\mathbb{R}^n$ and $\mathbb{R}^m$. Therefore, any vector $\mathbf{b} \in \mathbb{R}^n$ can be expanded in the basis formed by the columns of $\mathbf{U}$ (also known as the *left singular vectors* of $\mathbf{A}$) and any vector $\mathbf{x} \in \mathbb{R}^m$ can be expanded in the basis formed by the columns of $\mathbf{V}$ (also known as the *right singular vectors* of $\mathbf{A}$). The vectors for these expansions $\hat{\mathbf{b}}$ and $\hat{\mathbf{x}}$, are given by:

$$\hat{\mathbf{b}} = \mathbf{U}^T\mathbf{b} \qquad \& \qquad \hat{\mathbf{x}} = \mathbf{V}^T\mathbf{x}$$

Now, if the relation $\mathbf{b} = \mathbf{A}\mathbf{x}$ holds, then we can infer the following:

$$\begin{aligned} \mathbf{U}^T\mathbf{b} &= \mathbf{U}^T\mathbf{A}\mathbf{x} \\ \Rightarrow \qquad \hat{\mathbf{b}} &= \mathbf{U}^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)\mathbf{x} \qquad \Rightarrow \qquad \hat{\mathbf{b}} = \mathbf{\Sigma}\hat{\mathbf{x}} \end{aligned}$$

Thus, the SVD allows us to assert that every matrix is diagonal, so long as we choose the appropriate bases for the domain and range spaces.

How does this link in to the previous analysis of PCA? Consider the $n \times m$ matrix, $\mathbf{A}$, for which we have a singular value decomposition, $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. There is a theorem from linear algebra which says that the non-zero singular values of A are the square roots of the nonzero eigenvalues of $\mathbf{A}\mathbf{A}^T$ or $\mathbf{A}^T\mathbf{A}$. The former assertion for the case $\mathbf{A}^T\mathbf{A}$ is proven in the following way:

$$\begin{aligned} \mathbf{A}^T\mathbf{A} &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) \\ &= (\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) \\ &= \mathbf{V}(\mathbf{\Sigma}^T\mathbf{\Sigma})\mathbf{V}^T \end{aligned}$$

We observe that $\mathbf{A}^T\mathbf{A}$ is *similar* to $\mathbf{\Sigma}^T\mathbf{\Sigma}$, and thus it has the same eigenvalues. Since $\mathbf{\Sigma}^T\mathbf{\Sigma}$ is a square $(m \times m)$, diagonal matrix, the eigenvalues are in fact the diagonal entries, which are the squares of the singular values. Note that the nonzero eigenvalues of each of the covariance matrices, $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$ are actually identical.

It should also be noted that we have effectively performed an eigenvalue decomposition for the matrix, $\mathbf{A}^T\mathbf{A}$. Indeed, since $\mathbf{A}^T\mathbf{A}$ is symmetric, this is an orthogonal diagonalisation and thus the eigenvectors of $\mathbf{A}^T\mathbf{A}$ are the columns of $\mathbf{V}$. This will be important in making the practical connection between the SVD and and the PCA of matrix $\mathbf{X}$, which is what we will do next.

Returning to the original $m \times n$ data matrix, $\mathbf{X}$, let us define a new $n \times m$ matrix, $\mathbf{Z}$:

$$\mathbf{Z} = \frac{1}{\sqrt{n-1}}\mathbf{X}^T$$

Recall that since the m rows of $\mathbf{X}$ contained the n data samples, we subtracted the row average from each entry to ensure zero mean across the rows. Thus, the new matrix, $\mathbf{Z}$ has *columns* with zero mean. Consider forming the $m \times m$ matrix, $\mathbf{Z}^T\mathbf{Z}$:

$$\begin{aligned} \mathbf{Z}^T\mathbf{Z} &= \left(\frac{1}{\sqrt{n-1}}\mathbf{X}^T\right)^T\left(\frac{1}{\sqrt{n-1}}\mathbf{X}^T\right) \\ &= \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \\ \text{i.e.} \quad \mathbf{Z}^T\mathbf{Z} &= \mathbf{C}_\mathbf{X} \end{aligned}$$

We find that defining $\mathbf{Z}$ in this way ensures that $\mathbf{Z}^T\mathbf{Z}$ is equal to the covariance matrix of $\mathbf{X}$, $\mathbf{C}_\mathbf{X}$. From the discussion in the previous section, the principal components of $\mathbf{X}$ (which is what we are trying to identify) are the eigenvectors of $\mathbf{C}_\mathbf{X}$. Therefore, if we perform a singular value decomposition of the matrix $\mathbf{Z}^T\mathbf{Z}$, the principal components will be the columns of the orthogonal matrix, $\mathbf{V}$.

The last step is to relate the SVD of $\mathbf{Z}^T\mathbf{Z}$ back to the change of basis represented by equation (3):

$$\mathbf{Y} = \mathbf{P}\mathbf{X}$$

We wish to *project* the original data onto the directions described by the principal components. Since we have the relation $\mathbf{V} = \mathbf{P}^T$, this is simply:

$$\mathbf{Y} = \mathbf{V}^T\mathbf{X}$$

If we wish to recover the original data, we simply compute (using orthogonality of $\mathbf{V}$):

$$\mathbf{X} = \mathbf{V}\mathbf{Y}$$

5 Image Compression Using PCA

In the previous section, we developed a method for principal component analysis which utilised the singular value decomposition of an $m \times m$ matrix $\mathbf{Z}^T\mathbf{Z}$, where $\mathbf{Z} = \frac{1}{\sqrt{n-1}}\mathbf{X}^T$ and $\mathbf{X}$ was an $m \times n$ data matrix.

Since $\mathbf{Z}^T\mathbf{Z} \in \mathbb{R}^{m\times m}$, the matrix, $\mathbf{V}$ obtained in the singular value decomposition of $\mathbf{Z}^T\mathbf{Z}$ must also be of dimensions $m \times m$. Recall also that the columns of $\mathbf{V}$ are the principal component directions, and that the SVD automatically sorts these components in decreasing order of 'importance' or 'principality', so that the 'most principal' component is the first column of $\mathbf{V}$.

Suppose that before projecting the data using the relation, $\mathbf{Y} = \mathbf{V}^T\mathbf{X}$, we were to truncate the matrix, $\mathbf{V}$ so that we kept only the first $r < m$ columns. We would thus have a matrix $\tilde{\mathbf{V}} \in \mathbb{R}^{m\times r}$. The projection $\tilde{\mathbf{Y}} = \tilde{\mathbf{V}}^T\mathbf{X}$ is still dimensionally consistent, and the result of the product is a matrix, $\tilde{\mathbf{Y}} \in \mathbb{R}^{r\times n}$. Suppose that we then wished to transform this data back to the original basis by computing $\tilde{\mathbf{X}} = \tilde{\mathbf{V}}\tilde{\mathbf{Y}}$. We therefore recover the dimensions of the original data matrix, $\mathbf{X}$ and obtain, $\tilde{\mathbf{X}} \in \mathbb{R}^{m\times n}$.

The matrices, $\mathbf{X}$ and $\tilde{\mathbf{X}}$ are of the same dimensions, but they are not the same matrix, since we truncated the matrix of principal components $\mathbf{V}$ in order to obtain $\tilde{\mathbf{X}}$. It is therefore reasonable to conclude that the matrix, $\tilde{\mathbf{X}}$ has in some sense, 'less information' in it than the matrix $\mathbf{X}$. Of course, in terms of memory allocation on a computer, this is certainly not the case since both matrices have the same dimensions and would therefore allotted the same amount of memory. However, the matrix, $\tilde{\mathbf{X}}$ can be computed as the product of two smaller matrices ($\tilde{\mathbf{V}}$ and $\tilde{\mathbf{Y}}$). This, together with the fact that the 'important' information in the matrix is captured by the first principal components suggests a possible method for image compression.

During the subsequent analysis, we shall work with a standard test image that is often used in image processing and image compression. It is a greyscale picture of a butterfly, and is displayed in Figure 5. We will use MATLAB to perform the following analysis, though the principles can be applied in other computational packages.

Figure 5: The 'Butterfly' greyscale test image

MATLAB considers greyscale images as 'objects' consisting of two components, a matrix of pixels, and a colourmap. The 'Butterfly' image above is stored in a 512×512 matrix (and therefore has this number of pixels). The colourmap is a 512×3 matrix. For RGB colour images, each image can be stored as a single $512 \times 512 \times 3$ matrix, where the third dimension stores three numbers in the range $[0, 1]$ corresponding to each pixel in the 512×512 matrix,

representing the intensity of the red, green and blue components.

For a greyscale image such as the one we are dealing with, the colourmap matrix has three identical columns with a scale representing intensity on the one dimensional grey scale. Each element of the pixel matrix contains a number representing a certain intensity of grey scale for an individual pixel. MATLAB displays all of the 512×512 pixels simultaneously with the correct intensity and the greyscale image that we see is produced.
The 512×512 matrix containing the pixel information is our data matrix, X. We will perform a principal component analysis of this matrix, using the SVD method outlined above. The steps involved are exactly as described above and summarised in the following MATLAB code.

```
[fly,map] = imread('butterfly.gif');    % load image into MATLAB
fly=double(fly);                        % convert to double precision
image(fly),colormap(map);               % display image
axis off, axis equal
[m n]=size(fly);
mn = mean(fly,2);                       % compute row mean
X = fly - repmat(mn,1,n);               % subtract row mean to obtain X
Z=1/sqrt(n-1)*X';                       % create matrix, Z
covZ=Z'*Z;                              % covariance matrix of Z
%% Singular value decomposition
[U,S,V] = svd(covZ);
variances=diag(S).*diag(S);             % compute variances
bar(variances(1:30))                    % scree plot of variances
%% Extract first 20 principal components
PCs=40;
VV=V(:,1:PCs);
Y=VV'*X;                                % project data onto PCs
ratio=256/(2*PCs+1);                    % compression ratio
XX=VV*Y;                                % convert back to original basis
XX=XX+repmat(mn,1,n);                   % add the row means back on
image(XX),colormap(map),axis off;       % display results
```

Figure 6: MATLAB code for image compression PCA

In this case, we have chosen to use the first 40 (out of 512) principal components. What compression ratio does this equate to? To answer this question, we need to compare the amount of data we would have needed to store previously, with what we can now store. Without compression, we would still have our 512×512 matrix to store. After selecting the first 40 principal components, we have the two matrices $\tilde{\mathbf{V}}$ and $\tilde{\mathbf{Y}}$ (`VV` and `YY`) in the above MATLAB code) from which we can obtain a 512×512 pixel matrix by computing the matrix product.

Matrix $\tilde{\mathbf{V}}$ is 512×40, whilst matrix $\tilde{\mathbf{Y}}$ is 40×512. There is also one more matrix that we must use if we wish to display our image - the vector of means which we add back on after converting back to the original basis (this is just a 512×1 matrix which we can later copy into a larger matrix to add to $\tilde{\mathbf{X}}$). We therefore have reduced the number of columns needed from 512 to $40 + 40 + 1 = 41$ and the compression ratio is then calculated in the following way:

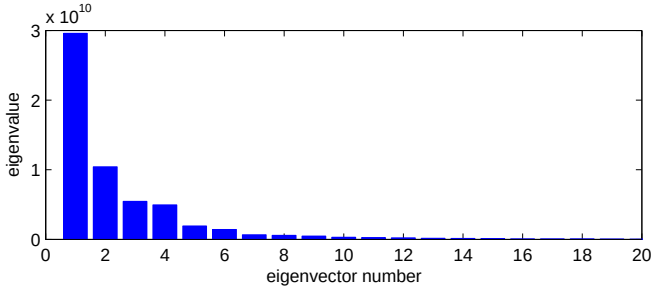
$$512 : 81 \qquad \text{i.e. approximately} \qquad \mathbf{6.3 : 1} \qquad \text{compression}$$

A decent ratio it seems, however what does the compressed image look like? The image for 40 principal components (**6.3:1** compression) is displayed in Figure 7.

Figure 7: 40 principal components (**6.3:1** compression)

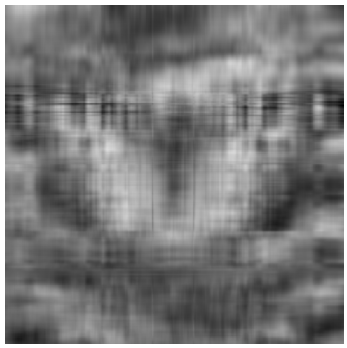
The loss in quality is evident (after all, this *lossy* compression, as opposed to *lossless* compression), however considering the compression ratio, the trade off seems quite good. Let's look next at the eigenspectrum, in Figure 8.

Figure 8: Eigenspectrum (first 20 eigenvalues)



The first principal component accounts for 51.6% of the variance, the first two account for 69.8%, the first six 93.8%. This type of plot is not so informative here, as accounting for 93.8% of the variance in the data does not correspond to us seeing a clear image, as is shown in Figure 9.

Figure 9: Image compressed using 6 principal components



On the next page, in Figure 5, a selection of images is shown with an increasing number of principal components retained. In Table 5, the cumulative sum of the contribution from the first 10 variances is displayed.

Eigenvector Number	Cumulative proportion of variance
1	0.5160
2	0.6979
3	0.7931
4	0.8794
5	0.9130
6	0.9378
7	0.9494
8	0.9596
9	0.9678
10	0.9732

Table 2: Cumulative variance accounted for by PCs

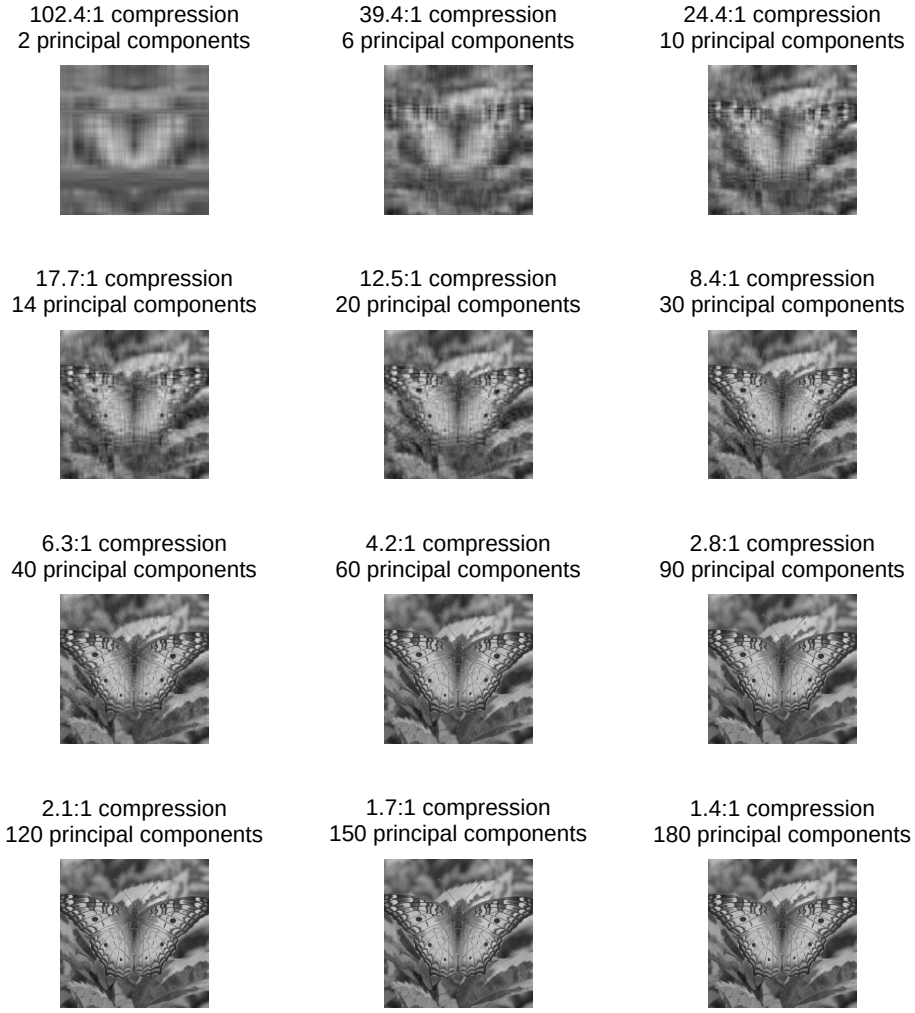


Figure 10: The visual effect of retaining principal components

6 Blind Source Separation

The final application of PCA in this report is motivated by the 'cocktail party problem', a diagram of which is displayed in Figure 11. Imagine we have N people at a cocktail party. The N people are all speaking at once, resulting in a mixture of all the voices. Suppose that we wish to obtain the individual monologues from this mixture - how would we go about doing this?

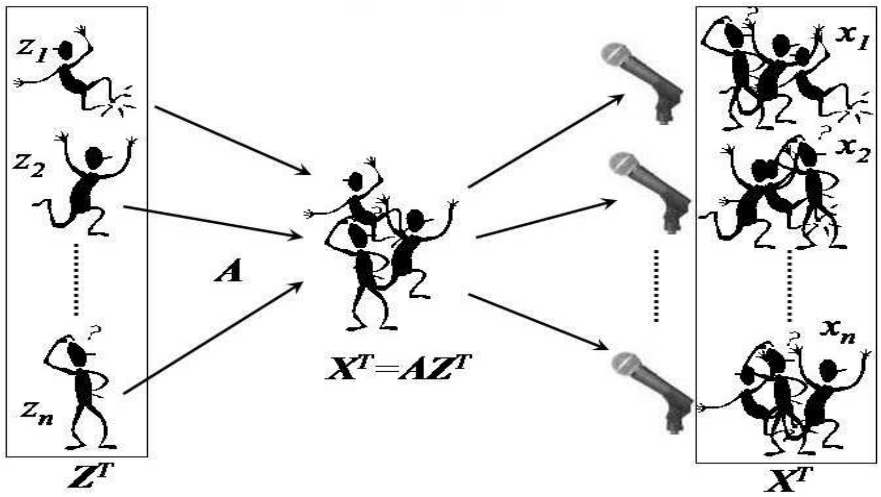


Figure 11: The cocktail party problem (image courtesy of Gari Clifford, MIT)

The room has been equipped with exactly n microphones, spread around at different points in the room. Each microphone thus records a slightly different version of the combined signal, together with some random noise. By analysing these n combined signals, suing PCA, it is possible to both de-noise the group signal, and to separate out the original sources. A formal statement of this problem is:

- Matrix $\mathbf{Z} \in \mathbb{R}^{m \times n}$ consists of m samples of n independent sources
- The signals are mixed together *linearly* using a matrix, $\mathbf{A} \in \mathbb{R}^{n \times n}$
- The matrix of observations is represented as the product $\mathbf{X}^T = \mathbf{A}\mathbf{Z}^T$
- We attempt to demix the observations by finding $\mathbf{W} \in \mathbb{R}^{n \times n}$ s.t. $\mathbf{Y}^T = \mathbf{W}\mathbf{X}^T$
- The hope is that $\mathbf{Y} \approx \mathbf{Z}$, and thus $\mathbf{W} \approx \mathbf{A}^{-1}$

These points reflect the assumptions of the blind source separation (BSS) problem:

1. The mixture of source signals must be linear
2. The source signals are *independent*
3. The mixture (the matrix $\mathbf{A}$) is stationary (constant)
4. The number of observations (microphones) is the same as the number of sources

In order to use PCA for BSS, we need to define *independence* in terms of the variance of the signals. In analogy with the previous examples and discussion of PCA in Section 3, we assume that we will be able to de-correlate the individual signals by finding the (orthogonal)

directions of maximal variance for the matrix of observations, $\mathbf{Z}$. It is therefore possible to again use the SVD for this analysis. Consider the 'skinny' SVD of $\mathbf{X} \in \mathbb{R}^{m\times n}$:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \qquad \text{where} \qquad \mathbf{U} \in \mathbb{R}^{m\times n}, \qquad \mathbf{\Sigma} \in \mathbb{R}^{n\times n}, \qquad \mathbf{V} \in \mathbb{R}^{n\times n}$$

Comparing the the skinny SVD with the full SVD, in both cases, the $n \times n$ matrix $\mathbf{V}$ is the same. Assuming that $m \geq n$, the diagonal matrix of singular values, $\mathbf{\Sigma}$, is square $(n \times n)$ in the skinny case, and rectangular $(m \times n)$ in the full case, with the additional $m - n$ rows being 'ghost' rows (i.e. have entries that are all zero). The first n columns of the matrix $\mathbf{U}$ in the full case are identical to the n columns of the skinny case $\mathbf{U}$, with the additional $m - n$ columns being arbitrary orthogonal appendments.

Recall that we are trying to approximate the original signals matrix ($\mathbf{Z} \approx \mathbf{Y}$) by trying to find a matrix ($\mathbf{W} \approx \mathbf{A}^{-1}$) such that:

$$\mathbf{Y}^T = \mathbf{W}\mathbf{X}^T$$

This matrix is obtained by rearranging the equation for the skinny SVD.

$$\begin{aligned} \mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad &\Rightarrow \quad \mathbf{X}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\ &\Rightarrow \quad \mathbf{U}^T = \mathbf{\Sigma}^{-T}\mathbf{V}^T\mathbf{X}^T \end{aligned}$$

Thus, we identify our approximation to the de-mixing matrix as $\mathbf{W} = \mathbf{\Sigma}^{-T}\mathbf{V}^T$, and our de-mixed signals are therefore the columns of the matrix $\mathbf{U}$. Note that since the matrix $\mathbf{\Sigma}$ is square and diagonal, $\mathbf{\Sigma}^{-T} = \mathbf{\Sigma}^{-1}$, which is computed by simply taking the reciprocal value of each diagonal entry of $\mathbf{\Sigma}$. However, if we are using the SVD method, it is not necessary to worry about explicitly calculating the matrix $\mathbf{W}$, since the SVD automatically delivers us the de-mixed signals.

To illustrate this, consider constructing the matrix $\mathbf{Z} \in \mathbb{R}^{2001\times 3}$ consisting of the following three signals (the columns) sampled at 2001 equispaced points on the interval $[0, 2000]$ (the rows).

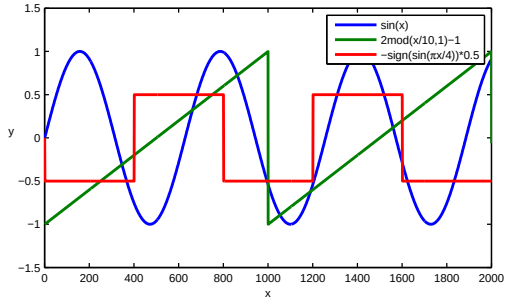


Figure 12: Three input signals for the BSS problem

We now construct a 3×3 mixing matrix, $\mathbf{A}$ by randomly perturbing the 3×3 identity matrix. The MATLAB code `A=round((eye(M)+0.01*randn(3,3))*1000)/1000` achieves this. To demonstrate, an example mixing matrix could be therefore be:

$$\mathbf{A} = \begin{pmatrix} 1.170 & -0.029 & 0.089 \\ -0.071 & 1.115 & -0.135 \\ -0.165 & 0.137 & 0.806 \end{pmatrix}$$

To simulate the cocktail party, we will also add some noise to the signals before the mix (for this, we use the MATLAB code `Z=Z+0.02*randn(2001,3)`). After adding this noise and mixing the signals to obtain $\mathbf{X}$ via $\mathbf{X}^T = \mathbf{A}\mathbf{Z}^T$, we obtain the following mixture of signals:

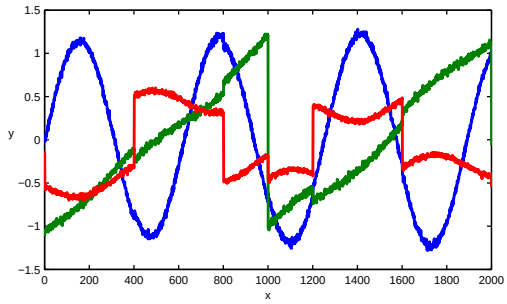


Figure 13: A mixture of the three input signals, with noise added

This mixing corresponds to each of the three microphones in the example above being close to a unique guest of the cocktail party, and therefore predominantly picking up what that individual is saying. We can now go ahead and perform a (skinny) singular value decomposition of the matrix, X, using the command, `[u,s,v]=svd(X,0)`. Figure 14 shows the separated signals (the columns of $\mathbf{U}$) plotted together, and individually.

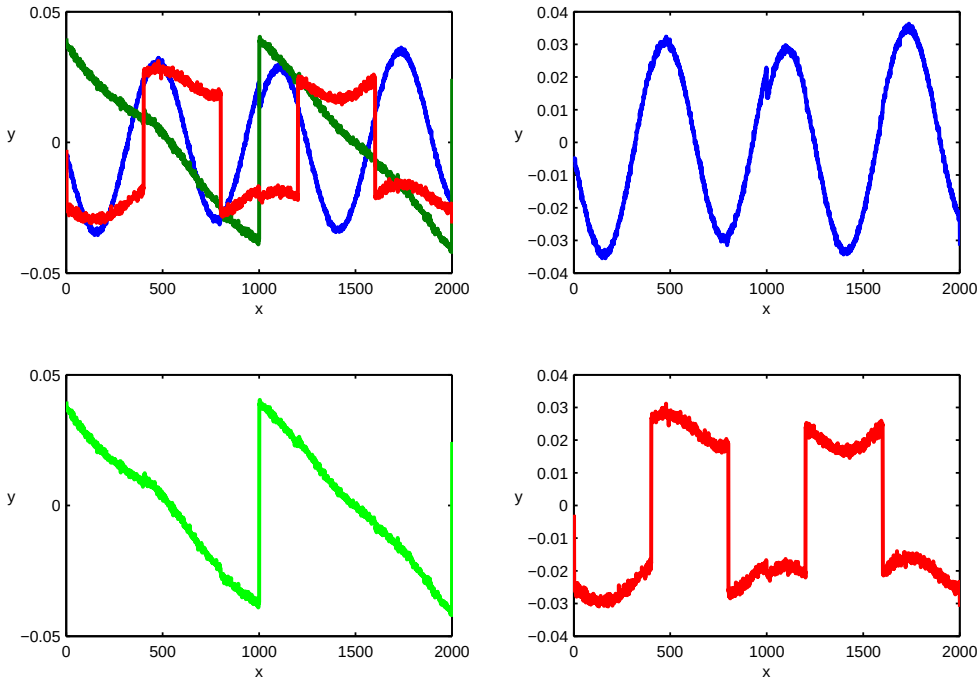


Figure 14: The separated signals plotted together, and individually

We observe that the extracted signals are quite easily identifiable representations the three input signals, however note that the sin and sawtooth functions have been inverted, and that the scaling is different to that of the inputted signals. The performance of PCA for blind source separation is good in this case because the mixing matrix was close to the identity. If we generate normally distributed random numbers for the elements of the mixing matrix, we can get good results, but we can also get poor results.

In the following figures, the upper left plot shows the three mixed signals and the subsequent 3 plots are the SVD extractions. Figures 15, 16 and 17 show examples of relatively good performance in extracting the original signals from the randomly mixed combination, whilst 18 shows a relatively poor performance.

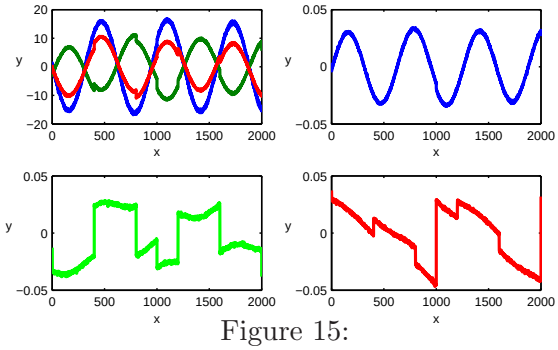


Figure 15:

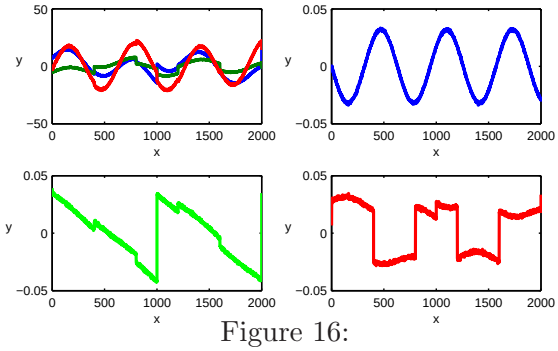


Figure 16:

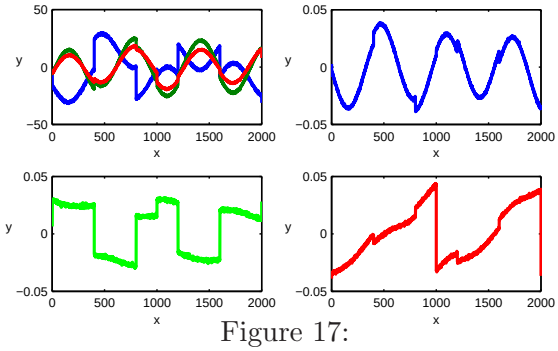


Figure 17:

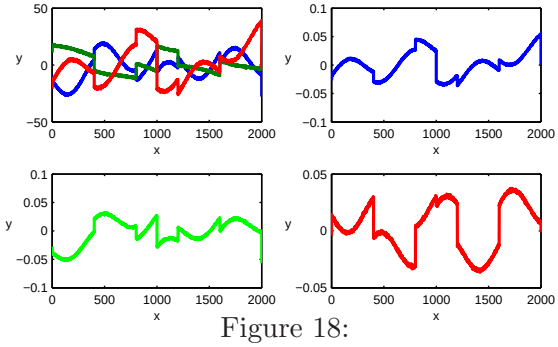


Figure 18:

7 Conclusions

My aim in writing this article was that somebody with a similar level of mathematical knowledge as myself (i.e. early graduate level) would be able to gain a good introductory understanding of PCA by reading this essay. I hope that they would understand that it is a diverse tool in data analysis, with many applications, three of which we have covered in detail here. I would also hope that they would gain an good understanding of the surrounding mathematics, and the close link that PCA has with the singular value decomposition.

I embarked upon writing this essay with only one application in mind, that of Blind Source Separation. However, when it came to researching the topic in detail, I found that there were many interesting applications of PCA, and I identified dimensional reduction in multivariate data analysis and image compression as being two of the most appealing alternative applications. Though it is a powerful technique, with a diverse range of possible applications, it is fair to say that PCA is not necessarily the best way to deal with each of the sample applications that I have discussed.

For the multivariate data analysis example, we were able to identify that the inhabitants of Northern Ireland were in some way different in their dietary habits to those of the other three countries in the UK. We were also able to identify particular food groups with the eating habits of Northern Ireland, yet we were limited in being able to make distinctions between the dietary habits of the English, Scottish and Welsh. In order to explore this avenue, it would perhaps be necessary to perform a similar analysis on just those three countries.

Image compression (and more generally, data compression) is by now getting to be a mature field ,and there are many sophisticated technologies available that perform this task. JPEG is an obvious and comparable example that springs to mind (JPEG can also involve lossy compression). JPEG utilises the discrete cosine transform to convert the image to a frequency-domain representation and generally achieves much higher quality for similar compression ratios when compared to PCA. Having said this, PCA is a nice technique in its own right for implementing image compression and it is nice to find such a pleasing implementation.

As we saw in the last example, Blind Source Separation can cause problems for PCA under certain circumstances. PCA will not be able to separate the individual sources if the signals are combined nonlinearly, and can produce spurious results even if the combination is linear. PCA will also fail for BSS if the data is non-Gaussian. In this situation, a well known technique that works is called Independent Component Analysis (ICA). The main philosophical difference between the two methods is that PCA defines independence using *variance*, whilst ICA defines independence using *statistical independence* - it identifies the principal components by maximising the statistical independence between each of the components.

Writing the theoretical parts of this essay (Sections 3 and 4) was a very educational experience and I was aided in doing this by the excellent paper by Jonathon Shlens, 'A Tutorial on Principal Component Analysis'[2], and the famous book on Numerical Linear Algebra by Lloyd N. Trefethen and David Bau III[4]. However, the original motivation for writing this special topic was from the excellent lectures in Signals Processing delivered by Dr. I Drobnjak and Dr. C. Orphinadou during Hilary term of 2008, at the Oxford University Mathematical Institute.

8 Appendix - MATLAB

Figure 19: MATLAB code: Data Analysis

```
X = [105 103 103 66; 245 227 242 267; 685 803 750 586;
    147 160 122 93; 193 235 184 209; 156 175 147 139;
    720 874 566 1033; 253 265 171 143; 488 570 418 355;
    198 203 220 187; 360 365 337 334; 1102 1137 957 674;
    1472 1582 1462 1494; 57 73 53 47; 1374 1256 1572 1506;
    375 475 458 135; 54 64 62 41];
covmatrix=X*X'; data = X; [M,N] = size(data); mn = mean(data,2);
data = data - repmat(mn,1,N); Y = data' / sqrt(N-1); [u,S,PC] = svd(Y);
S = diag(S); V = S .* S; signals = PC' * data;
plot(signals(1,1),0,'b.',signals(1,2),0,'b.',...
    signals(1,3),0,'b.',signals(1,4),0,'r.','markersize',15)
xlabel('PC1')
text(signals(1,1)-25,-0.2,'Eng'),text(signals(1,2)-25,-0.2,'Wal'),
text(signals(1,3)-20,-0.2,'Scot'),text(signals(1,4)-30,-0.2,'N Ire')
plot(signals(1,1),signals(2,1),'b.',signals(1,2),signals(2,2),'b.',...
signals(1,3),signals(2,3),'b.',signals(1,4),signals(2,4),'r.',...
    'markersize',15)
xlabel('PC1'),ylabel('PC2')
text(signals(1,1)+20,signals(2,1),'Eng')
text(signals(1,2)+20,signals(2,2),'Wal')
text(signals(1,3)+20,signals(2,3),'Scot')
text(signals(1,4)-60,signals(2,4),'N Ire')

plot(PC(1,1),PC(1,2),'m.',PC(2,1),PC(2,2),'m.',...
     PC(3,1),PC(3,2),'m.',PC(4,1),PC(4,2),'m.',...
     PC(5,1),PC(5,2),'m.',PC(6,1),PC(6,2),'m.',...
     PC(7,1),PC(7,2),'m.',PC(8,1),PC(8,2),'m.',...
     PC(9,1),PC(9,2),'m.',PC(10,1),PC(10,2),'m.',...
     PC(11,1),PC(11,2),'m.',PC(12,1),PC(12,2),'m.',...
     PC(13,1),PC(13,2),'m.',PC(14,1),PC(14,2),'m.',...
     PC(15,1),PC(15,2),'m.',PC(16,1),PC(16,2),'m.',...
     PC(17,1),PC(17,2),'m.','markersize',15)

xlabel('effect(PC1)'),ylabel('effect(PC2)')

text(PC(1,1),PC(1,2)-0.1,'Cheese'),text(PC(2,1),PC(2,2)-0.1,'Carcass meat')
text(PC(3,1),PC(3,2)-0.1,'Other meat'),text(PC(4,1),PC(4,2)-0.1,'Fish')
text(PC(5,1),PC(5,2)-0.1,'Fats and oils'),text(PC(6,1),PC(6,2)-0.1,'Sugars')
text(PC(7,1),PC(7,2)-0.1,'Fresh potatoes')
text(PC(8,1),PC(8,2)-0.1,'Fresh Veg')
text(PC(9,1),PC(9,2)-0.1,'Other Veg')
text(PC(10,1),PC(10,2)-0.1,'Processed potatoes')
text(PC(11,1),PC(11,2)-0.1,'Processed Veg')
text(PC(12,1),PC(12,2)-0.1,'Fresh fruit'),
text(PC(13,1),PC(13,2)-0.1,'Cereals'),text(PC(14,1),PC(14,2)-0.1,'Beverages')
text(PC(15,1),PC(15,2)-0.1,'Soft drinks'),
text(PC(16,1),PC(16,2)-0.1,'Alcoholic drinks')
text(PC(17,1),PC(17,2)-0.1,'Confectionery')
%%
bar(V)
xlabel('eigenvector number'), ylabel('eigenvalue')
%%
t=sum(V);cumsum(V/t)
```

Figure 20: MATLAB code : Image Compression

```
clear all;close all;clc,

[fly,map] = imread('butterfly.gif');
fly=double(fly);
whos

image(fly)
colormap(map)
axis off, axis equal

[m n]=size(fly);
mn = mean(fly,2);
X = fly - repmat(mn,1,n);

Z=1/sqrt(n-1)*X';
covZ=Z'*Z;

[U,S,V] = svd(covZ);

variances=diag(S).*diag(S);
bar(variances,'b')
xlim([0 20])
xlabel('eigenvector number')
ylabel('eigenvalue')

tot=sum(variances)
[[1:512]' cumsum(variances)/tot]

PCs=40;
VV=V(:,1:PCs);
Y=VV'*X;
ratio=512/(2*PCs+1)

XX=VV*Y;

XX=XX+repmat(mn,1,n);

image(XX)
colormap(map)
axis off, axis equal

z=1;
for PCs=[2 6 10 14 20 30 40 60 90 120 150 180]
    VV=V(:,1:PCs);
    Y=VV'*X;
    XX=VV*Y;
    XX=XX+repmat(mn,1,n);
    subplot(4,3,z)
    z=z+1;
    image(XX)
    colormap(map)
    axis off, axis equal
    title({[num2str(round(10*512/(2*PCs+1))/10) ':1 compression'];...
        [int2str(PCs) ' principal components']})
end
```

Figure 21: MATLAB code : Blind Source Separation

```
clear all; close all; clc;

set(0,'defaultfigureposition',[40 320 540 300],...
'defaultaxeslinewidth',0.9,'defaultaxesfontsize',8,...
'defaultlinelinewidth',1.1,'defaultpatchlinewidth',1.1,...
'defaultlinemarkersize',15), format compact, format short

x=[0:0.01:20]';
signalA = @(x) sin(x);
signalB = @(x) 2*mod(x/10,1)-1;
signalC = @(x) -sign(sin(0.25*pi*x))*0.5;
Z=[signalA(x) signalB(x) signalC(x)];

[N M]=size(Z);
Z=Z+0.02*randn(N,M);
[N M]=size(Z);
A=round(10*randn(3,3)*1000)/1000
X0=A*Z';
X=X0';

figure
subplot(2,2,1)
plot(X,'LineWidth',2)
xlim([0,2000])
xlabel('x'),ylabel('y','Rotation',0)

[u,s,v] = svd(X,0);

subplot(2,2,2)
plot(u(:,1),'b','LineWidth',2)
xlim([0,2000])
xlabel('x'),ylabel('y','Rotation',0)

subplot(2,2,3)
plot(u(:,2),'g','LineWidth',2)
xlim([0,2000])
xlabel('x'),ylabel('y','Rotation',0)

subplot(2,2,4)
plot(u(:,3),'r','LineWidth',2)
xlim([0,2000])
xlabel('x'),ylabel('y','Rotation',0)
```

References

[1] UMETRICS
Multivariate Data Analysis
http://www.umetrics.com/default.asp/pagename/methods_MVA_how8/c/1#

[2] Jonathon Shlens
A Tutorial on Principal Component Analysis
http://www.brainmapping.org/NITP/PNA/Readings/pca.pdf

[3] Soren Hojsgaard
Examples of multivariate analysis Principal component analysis (PCA)
Statistics and Decision Theory Research Unit, Danish Institute of Agricultural Sciences

[4] Lloyd Trefethen & David Bau
Numerical Linear Algebra
SIAM

[5] Signals and Systems Group
Uppsala University
http://www.signal.uu.se/Courses/CourseDirs/...
...DatoriseradMI/DatoriseradMI05/instrPCAlena.pdf

[6] Dr. I. Drobnjak
Oxford University
MSc MMSC Signals Processing Lecture Notes (PCA/ICA)

[7] Gari Clifford
MIT
Blind Source Separation: PCA & ICA